A Small, Low Cost Event-Driven Architecture for Spiking Neural Networks on FPGAs

J. Parker Mitchell
mitchelljp1@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Catherine D. Schuman
schumancd@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Thomas E. Potok
potokte@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT

Currently, there is a lack of availability of low cost, low power neuromorphic hardware. In this work, we introduce the μCaspian architecture along with an associated development PCB design which provides a low cost and SWaP (size, weight, and power) optimized neuromorphic hardware platform. Further, our proposed system only uses commercial off the shelf components and an open source FPGA workflow to maximize the accessibility of μCaspian to all researchers.

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; **Reconfigurable computing**.

KEYWORDS

neuromorphic, reconfigurable computing, spiking neural network, fpga

ACM Reference Format:

J. Parker Mitchell, Catherine D. Schuman, and Thomas E. Potok. 2020. A Small, Low Cost Event-Driven Architecture for Spiking Neural Networks on FPGAs. In *International Conference on Neuromorphic Systems 2020 (ICONS 2020), July 28–30, 2020, Oak Ridge, TN, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3407197.3407216

1 INTRODUCTION

There has been a significant increase in interest in the field of neuromorphic computing in recent years for a variety of reasons, including the looming end of Moore's law and the end of Dennard scaling, as well as the rise of success of artificial intelligence algorithms and deep learning. In order for neuromorphic computing to be successful as a computing platform in the coming years, we need to begin to establish a community of users who have access to both hardware and software systems. One barrier to establishing this community is the lack of availability of hardware. Though there is a tremendous amount of research in the development of new neuromorphic hardware, as well as the development of neuromorphic chips like IBM's TrueNorth and Intel's Loihi, it is still extremely difficult for a non-neuromorphic researcher to get access to utilize neuromorphic hardware. Moreover, specifically for academic groups developing neuromorphic hardware, it can be extremely difficult and/or expensive to fabricate custom neuromorphic hardware systems, which has further decreased the availability and accessibility of these systems.

With this in mind, we have developed a neuromorphic development platform called Caspian, which has both software and hardware components [5]. The initial hardware platform for Caspian is based on field programmable gate arrays (FPGAs), because they are inexpensive (allowing for more accessibility), and they can be programmed and updated as new neuromorphic research emerges. In this work, we present the μCaspian development board, which is based on a very small, inexpensive, energy efficient FPGA. This particular platform was developed with several different use cases in mind. First, because of its size and energy usage, it is well-suited for edge deployment applications. Second, because it is inexpensive and has an associated user-friendly software development system in Python for programming the system, it is also amenable for educational purposes.

2 BACKGROUND AND RELATED WORK

Due to the slow and expensive fabrication processes associated with developing new hardware, field programmable gate array (FPGA)-based neuromorphic implementations have been common in neuromorphic computing [7]. Because they are easily available and relatively inexpensive, FPGAs are convenient both as a prototype for a potential future custom chip design, but also as an end-design for a low cost, off-the-shelf implementation that can give some of the energy efficiency and/or speed benefits of custom hardware. In recent years, energy efficent neuromorphic implementations on FPGAs have been used to do real-time cortical simulations [10], to accelerate training [9], and controlling the locomotion of a multi-legged robot [4]. These types of approaches are targeted towards relatively large FPGAs, where the goal of this work is to provide a very inexpensive, very energy efficient platform for rapid development and deployment, specifically for edge applications.

There are some example neuromorphic development platforms that have a complete workflow, including both hardware and software components. Notably, the Nengo [2] software framework, built on principles of the Neural Engineering Framework, has several supported hardware backends, including an FPGA-based approach [6].

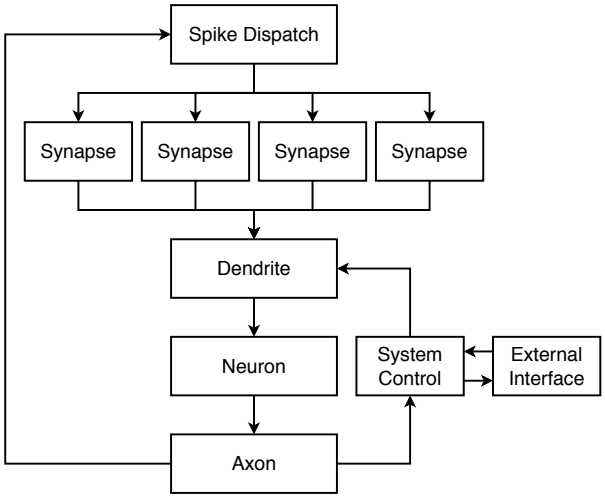

Figure 1: μCaspian Architecture Overview

A key issue with the Nengo-based approach is that it can be difficult or impractical to use outside of the context of the Neural Engineering Framework.

Other neuromorphic development platforms include commercial systems such as Intel's Loihi [3] and IBM's TrueNorth [1]. These systems have associated software to aid in development; however, the hardware systems themselves are not readily available to the community, which can make it difficult or impossible to deploy or evaluate on a real-world application.

3 ARCHITECTURE

μCaspian implements an event-driven pipeline of neuromorphic modules. As shown in Figure 1, the architecture is divided into a few major components – spike dispatch, synapses, dendrites, neurons, axons, and system control. Each component operates as a separate stage of the overall pipeline with a simple flow control interface and global time synchronization. Any associated configuration or state is stored local to the component using small block RAMs. This design allows the implementation to be both flexible and modular. In the following subsections, we explain the role of each component.

3.1 System Control and Communication

The system controller is responsible for processing input packets and generating output packets. It also handles control and synchronization signals for the overall system. The system is not allowed to progress to the next time step until every module in the system reports as idle after which time is incremented and a sync pulse is sent.

For both input and output, μCaspian implements a variable length packet scheme to minimize required bandwidth. Input packets may:

- Add charge to a specified dendrite/neuron
- Advance network time
- Clear the current state of the network
- Clear the configuration of the network
- Set the configuration values for a neuron or synapse

Some input input packets (clear, configuration) generate return acknowledgement packets upon completion of the operation. μCaspian also provides the spiking activity of neurons through output packets. Each neuron can be configured to send its output as needed.

3.2 Synapse

In μCaspian, there are four synapse units. Each synapse unit contains the configuration of 1024 synapses including its signed 8-bit weight value and its target, or post-synaptic, neuron. When there is an incoming spike, the synapse is responsible for passing the synaptic weight to correct dendrite address. In the future, it is possible to implement additional functionality into this module for features like synaptic plasticity.

3.3 Dendrite

The dendrite is where intermediate charge values are accumulated during the course of a single time step. Within each time step, it receives input charge from synapses as well as input charge from the system controller. The dendrite accumulates charge into a block RAM where the address corresponds to the target neuron id. There are two separate blocks RAMs which alternate on every time step, so at any one point in time, charge is being written to one RAM while the other RAM is flushing its charge to the neuron component. At the end of each time step, the RAMs swap roles such that the previously accumulated charge will then get flushed to the neurons. This functionality allows for coherent and predictable operation by avoiding any race conditions or requiring duplicate checking of neurons state.

3.4 Neuron

The neuron provides long term storage of charge. Every time step, charge from one of the dendrite buffers is flushed. Each neuron's charge value is updated as necessary, and if the charge exceeds the configured threshold, the neuron will emit a spike to the axon and reset the charge value back to zero. A neuron may be configured with an 8-bit unsigned threshold, and each neuron has a 16-bit signed charge. Neurons are allowed to hold a negative charge up to -2^{15}, but any positive charge of 2^8 or greater will cause a spike because the maximum configurable threshold is currently 255.

3.5 Axon

The axon serves to map spikes from neurons to the appropriate range of synapses. All output synapses for a given neuron are allocated to a contiguous range of synapse addresses. This means the mapping of a neuron's spike to synapses can be stored by the first index and the total number of synapses. Axons also provide a temporal delay capability which shifts the timing of each spike from a given neuron. Each axon can be configured with a separate delay value from 0 to 15 time steps. Delay is implemented using virtual shift registers where the state is stored as a bit field in a block RAM. New spikes are added as the i-th bit where i is the number of cycles of delay required, and the bit field is shifted once per time step. After shifting, if the least significant bit is set, the axon emits the spike at that time.

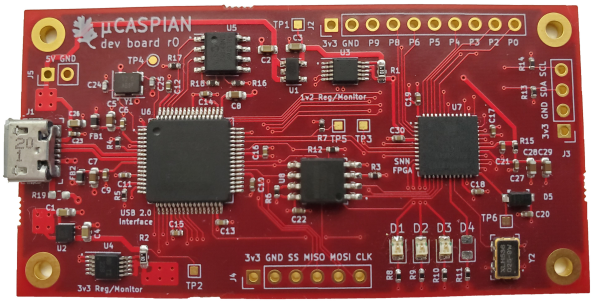

Figure 2: µCaspian Development Board

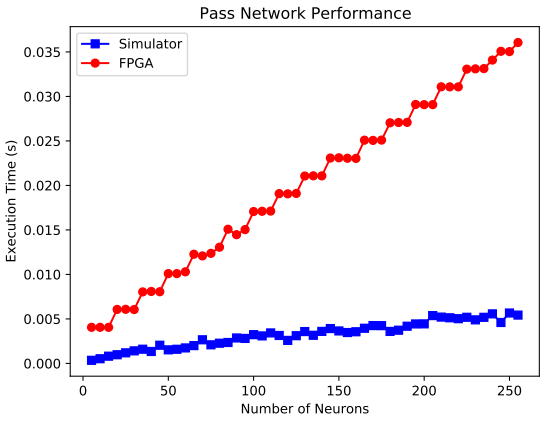

Figure 3: A performance comparison using length pass networks of varying length up to the device maximum of 256

3.6 Spike Dispatch

The spike dispatch is an intermediary module between the axon and the synapse units which coordinates the four synapse units. It iterates across the synapses as dictated by the output from the axon while decoupling the axon from the exact configuration of the synapses.

4 DEVELOPMENT BOARD

As a development platform for the µCaspian architecture, we developed a custom PCB with a Lattice ice40 UltraPlus FPGA. The primary communication interface is USB 2.0 High Speed through an FTDI 2232H interface chip. This manages USB and bridges it to an asynchronous FIFO-style interface which connects to the FPGA. The FTDI chip also handles configuring the FPGA logic through SPI. It should be noted that USB 2.0 provides great usability for development, but it also incurs a significant latency overhead of at least 1ms which causes significant overhead on small transfers. To determine energy efficiency, the board includes power monitoring capabilities with two Texas Instruments INA226 chips which communicate through an I2C bus. The FPGA may be used as the I2C bus master, or an external microcontroller can connect through the pin header on the right side of the board. The resulting board is pictured in Figure 2.

The Lattice ice40 UltraPlus series was chosen specifically for a few key reasons. First, it offers favorable power characteristics with both low static and dynamic power with total power consumption on order of 10mW. Second, it offers an appropriate amount of logic and small block RAM resources for the proposed architecture. Third, there is an open source toolchain consisting of yosys and nextpnr [8] which can allow end users to modify and update the FPGA logic without downloading large software packages or purchasing licenses.

Currently, we are able to implement µCaspian with 256 neurons and 4096 synapses on this platform. There are no configuration restrictions in terms of network connectivity, so while this is a modest number of resources, it is quite useful for sparse, recurrent graph structures.

5 RESULTS

5.1 Pass Network

Evaluating passthrough networks is a simple benchmark task to compare µCaspian on an FGPA to the Caspian simulator. A pass network is a chain of neurons where each neuron has one incoming and one outgoing connection such that a spike will start a neuron 0 and eventually reach neuron $N-1$ where N is the size of the network. Each neuron has a minimal threshold such that any positive charge results in a spike. We evaluated pass networks from 5 to 255 neurons and applied 1000 input spikes. We evaluated the networks until we received 1000 output spikes which results in $N+1000$ time steps. The results of this experiment are shown in Figure 3. Across the tests, the FPGA is typically around 6 to 7 times slower than the simulator running on an Intel i7 5820K CPU. While this initially may seem less than ideal, this is actually a very compelling result from an energy efficiency standpoint. The FPGA is using on the order of 1000 times less power than the Intel CPU while running on the order of 10 times slower. Further, there are several planned implementation improvements which will yield improved results. In future work, we plan to carefully examine and quantify the energy and performance trade-offs between CPU-based simulation and FPGA evaluation.

5.2 All to All Recurrent Network

The next benchmark uses a fully connected recurrent network. Every neuron in the network is connected to each other neuron with a synapse of random weight. Every neuron is specified as an input and output node in the network. The rate of input for each neuron is randomly selected from a uniform distribution of 0 to 100 which corresponds to the interval between subsequent inputs. The networks are evaluated for 1000 time steps, and the results from 5 different random seeds are averaged to generate the final metric. The collected results are shown in Figure 4. Similar to the pass network tests, the FPGA is slower than the software simulator running on a workstation. In this test, the FPGA is typically between 3 to 6 times slower than the simulator, but again, it is important to

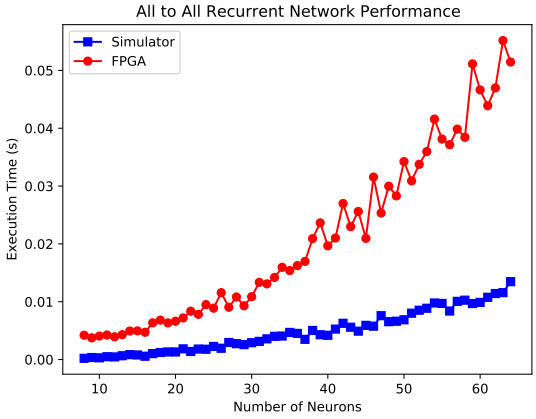

Figure 4: A performance comparison using all to all recurrent networks of varying size

remember that this still yields a many times over energy efficiency advantage for the FPGA based solution.

6 DISCUSSION AND FUTURE WORK

A key advantage of the proposed work is the utility of a low cost, low power neuromorphic platform using only off the shelf components. The system is intended to be simple to use by integrating with both a C++ and a Python API as well as a software-based simulation engine. However, a downside of our approach is that capacity is currently limited to relatively small networks and the energy efficiency of an FPGA solution will always be at a disadvantage compared to custom ASICs. Additionally, our development board uses a relatively slow clock speed which results in lower performance than simulation on a workstation but will offer much greater energy efficiency.

For future work, we intend to investigate the use and deployment of Caspian for a wide variety of different applications, including robotics, autonomous vehicles, and real-time sensor data analysis. Though this work presents a full development board, there are also opportunities to utilize μCaspian without the full board by embedding a small FPGA into the target application. The platform currently uses USB to communicate with PCs, but the interface can be exchanged with microcontroller friendly options like UART. In the future, we plan to measure the logic power consumption of the FPGA and also quantify the overhead of USB 2.0 based communication. These results are particularly relevant if the end user decides to embed μCaspian without the full development board.

This work focuses on the development specifically of the μCaspian system. However, the Caspian platform is scalable and can be extended for larger FPGAs for different use cases. The event-driven pipeline concept can extend to a larger number of neurons and synapses. Further, we can envision a multi-core approach where the cores execute subsets of a larger network or execute a collection of small individual networks.

Because one of the key motivations for the development of this work is to make neuromorphic computing more accessible for new users, we intend to develop learning modules for Caspian that include both software and hardware components. In particular, we hope to develop learning modules that will be appropriate for short workshops to introduce new users to neuromorphic computing, as well as longer, more detailed curricula that are appropriate for use in an academic setting (perhaps as part of a course on neuromorphic computing).

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725, and by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory.

REFERENCES

[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. 2015. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 10 (2015), 1537–1557.

[2] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. 2014. Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics* 7, 48 (2014). https://doi.org/10.3389/fninf.2013.00048

[3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.

[4] Erick Israel Guerra-Hernandez, Andres Espinal, Patricia Batres-Mendoza, Carlos Hugo Garcia-Capulin, Rene De J Romero-Troncoso, and Horacio Rostro-Gonzalez. 2017. A FPGA-based neuromorphic locomotion system for multi-legged robots. *IEEE Access* 5 (2017), 8301–8312.

[5] J Parker Mitchell, Catherine D Schuman, Robert M Patton, and Thomas E Potok. 2020. Caspian: A Neuromorphic Development Platform. In *Proceedings of the Neuro-inspired Computational Elements Workshop*. 1–6.

[6] Benjamin Morcos. 2019. *NengoFPGA: an FPGA Backend for the Nengo Neural Simulator*. Master's thesis. University of Waterloo.

[7] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. 2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963* (2017).

[8] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanovic. 2019. Yosys+ nextpnr: an open source framework from verilog to bitstream for commercial fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 1–4.

[9] Qian Wang, Youjie Li, Botang Shao, Siddhartha Dey, and Peng Li. 2017. Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA. *Neurocomputing* 221 (2017), 146–158.

[10] Runchun M Wang, Chetan S Thakur, and André van Schaik. 2018. An FPGA-based massively parallel neuromorphic cortex simulator. *Frontiers in neuroscience* 12 (2018), 213.